Rami Ilo
February, 10th, 2021

Exercise 2 | TKO_2096 Application of Data Analysis 2021

Prediction of the metal ion content from multi-parameter data

- Use K-Nearest Neighbor Regression with euclidean distance to predict total metal concentration (c_total), concentration of Cadmium (Cd) and concentration of Lead (Pb), for each sample using number of neighbors k = 3.

  - You may use Nearest Neighbor Regression from https://scikit-learn.org/stable/modules/neighbors.html
  - The data should be standarized using z-score.
  - Implement your own Leave-One-Out cross-validation and calculate the C-index for each output (c_total, Cd, Pb).
  - Implement your own Leave-Replicas-Out cross-validation and calculate the C-index for each output (c_total, Cd, Pb).
  - Return your solution as a Jupyter Notebook file (include your full name in the file name).
  - Submit to moodle your solution on **Wednesday 10 of February** at the latest.

Import libraries

In [1]:
```
#In this cell import all libraries you need. For example:
import numpy as np
import pandas as pd
from collections import Counter
from sklearn import neighbors
```

Read and visualize the dataset

In [2]:
```
#In this cell read the file Water_data.csv
df = pd.read_csv(r"G:\Dropbox\Applications of data analysis\Water_data.csv")
#Print the dataset dimesions (i.e. number of rows and columns)
df.info()
#Print the first 5 rows of the dataset
df.head(5)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 225 entries, 0 to 224
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   c_total  225 non-null    int64
 1   Cd       225 non-null    float64
 2   Pb       225 non-null    float64
 3   Mod1     225 non-null    int64
 4   Mod2     225 non-null    int64
 5   Mod3     225 non-null    int64
dtypes: float64(2), int64(4)
memory usage: 10.7 KB
```

Out[2]:

	c_total	Cd	Pb	Mod1	Mod2	Mod3
0	2000	800.0	1200.0	126430	2604	6996
1	35	14.0	21.0	20597	271	138677
2	35	14.0	21.0	24566	269	161573
3	35	35.0	0.0	105732	971	132590
4	100	20.0	80.0	57774	5416	93798

To show understanding of the data, answer the following questions:

- How many different mixtures of Cadmium (Cd) and Lead (Pb) were measured?
- How many total concentrations (c_total) were measured?
- How many mixtures have less than 4 replicas?
- How many mixtures have 4 or more replicas? Print out c_total, Cd and Pb for those concentrations.

In [3]:
```
#In this cell write the code to answer the previous questions and print the answers.

#Counter for num of duplicates/replicas
c = Counter(list(zip(df.c_total, df.Cd, df.Pb)))

print("There were {0} different mixtures of Cadmium (Cd) and Lead (Pd)".format(len(c)))
print("A total of {0} concentrations were measured".format(df["c_total"].nunique()))

less_than_4_dups = {i: j for i, j in c.items() if j < 4}
at_least_4_dups = {i: j for i, j in c.items() if j >= 4}

print("{0} Mixtures have less than {1} replicas".format(len(less_than_4_dups), 4))
print("{0} Mixtures have {1} or more replicas: ".format(len(at_least_4_dups), 4))
print(at_least_4_dups)
```

```
There were 67 different mixtures of Cadmium (Cd) and Lead (Pd)
A total of 12 concentrations were measured
43 Mixtures have less than 4 replicas
24 Mixtures have 4 or more replicas:
{(100, 20.0, 80.0): 4, (50, 40.0, 10.0): 4, (100, 100.0, 0.0): 4, (50, 0.0, 50.0): 4, (70, 0.0, 70.0): 4, (70, 56.0, 14.0): 4, (200, 160.0,
40.0): 4, (50, 50.0, 0.0): 4, (100, 0.0, 100.0): 4, (100, 40.0, 60.0): 4, (70, 42.0, 28.0): 4, (50, 30.0, 20.0): 4, (100, 80.0, 20.0): 4,
(200, 80.0, 120.0): 4, (70, 14.0, 56.0): 4, (200, 120.0, 80.0): 4, (200, 40.0, 160.0): 4, (200, 200.0, 0.0): 4, (200, 0.0, 200.0): 4, (70,
28.0, 42.0): 4, (50, 10.0, 40.0): 4, (70, 70.0, 0.0): 4, (100, 60.0, 40.0): 4, (50, 20.0, 30.0): 4}
```

Standardization of the dataset

In [4]:
```
#Standardize the dataset features by removing the mean and scaling to unit variance.
#In other words, use z-score to scale the dataset features (Mod1, Mod2, Mod3)
#Print the 5 first samples (i.e. rows) of the scaled dataset

#A function to calculate z-score for a column
def get_z_score(column):
    mean = np.mean(column)
    std = np.std(column)
    z_score = (column - mean) / (std)
    return z_score

df["Mod1"], df["Mod2"], df["Mod3"] = get_z_score(df["Mod1"]), get_z_score(df["Mod2"]), get_z_score(df["Mod3"])

df.head(5)
```

Out[4]:

	c_total	Cd	Pb	Mod1	Mod2	Mod3
0	2000	800.0	1200.0	0.166505	-0.508756	-1.499041
1	35	14.0	21.0	-0.892616	-0.701641	0.685861
2	35	14.0	21.0	-0.852896	-0.701806	1.065760
3	35	35.0	0.0	-0.040629	-0.643767	0.584863
4	100	20.0	80.0	-0.520568	-0.276268	-0.058789

C-index code

In [5]:
```
def cindex(true_labels, pred_labels):
    """Returns C-index between true labels and predicted labels"""

    counter = 0
    pairs = 0

    for i in range(0, len(true_labels)):
        assert len(true_labels) == len(pred_labels)
        #if negative label, test that labels corresponding prediction against all positive
        #labels' predictions
        x = true_labels[i]
        for j in range(0, len(true_labels)):
            if(true_labels[j] > x):
                pairs+=1
                if(pred_labels[i] < pred_labels[j]):
                    counter+=1
                elif(pred_labels[i] == pred_labels[j]):
                    counter+=0.5

    cindx = counter / pairs
    return cindx
```

In [6]:
```
#test cindex function with following values
true_labels = [-1, 1, 1, -1, 1]
predictions = [0.60, 0.80, 0.75, 0.75, 0.70]
cindx = cindex(true_labels, predictions)
print(cindx)
```

```
0.75
```

Functions

In [7]:

```
#Include here all the functions that you need to run in the data analysis part.
X = df[["Mod1", "Mod2", "Mod3"]].to_numpy()
y = df[["c_total", "Cd", "Pb"]]

#Map different concentrations and the indices of each set
def map_replicates(df):


    df = (df.groupby(df.columns.tolist()).apply(lambda x: tuple(x.index)).reset_index(name="indices"))
    return df

def loocv(X, y, n_neighbors, title):
    scores = []
    true_labels = []

    for i in range(0, len(X)):
        #Split the data: in every loop, everything but i:th instance is chosen as training data.
        (train_data, val_data, train_labels, val_labels) = np.delete(X, i, axis=0), X[i], np.delete(y, i, axis=0), y[i]

        knn = neighbors.KNeighborsRegressor(n_neighbors)
        knn.fit(train_data, train_labels)
        #Predict on the i:th element. 2D array is required
        val_pred = knn.predict([val_data])

        #Collect the predictions and correct labels in their respective arrays
        scores = np.append(scores, np.array(val_pred))
        true_labels = np.append(true_labels, np.array(val_labels))

    scores = np.array(scores)
    true_labels = np.array(true_labels)

    #Get the score from the 1...N predictions
    print("C-index score for {1}: {0}]".format(cindex(true_labels, scores), title))

def lrocv(X, y, yd, n_neighbors, title):
    scores = []
    true_labels = []

    #loop for the amount of different replicas/duplicates
    for i in range(0, len(yd)):

        #find the different indexes of the i:th unique element
        ith_rows = yd.iloc[i]["indices"]

        #construct the test set if index=ith_rows ie. duplicate with current elements, otherwise training set
        (train_data, val_data, train_labels, val_labels) = np.delete(X, ith_rows, axis=0), [X[j] for j in ith_rows],
np.delete(y, ith_rows, axis=0), [y[j] for j in ith_rows]

        knn = neighbors.KNeighborsRegressor(n_neighbors)
        knn.fit(train_data, train_labels)
        val_pred = knn.predict(val_data).flatten()

        scores = np.append(scores, np.array(val_pred).flatten())
        true_labels = np.append(true_labels, np.array(val_labels).flatten())

    scores = np.array(scores)
    true_labels = np.array(true_labels)

    print("C-index score for {1}: {0}]".format(cindex(true_labels, scores), title))
```

Results for Leave-One-Out cross-validation

In [8]:
```
#In this cell run your code for leave-One-Out cross-validation and print the corresponding results.


loocv(X, y["c_total"].to_numpy(), 3, "c_total")
loocv(X, y["Cd"].to_numpy(), 3, "Cd")
loocv(X, y["Pb"].to_numpy(), 3, "Pd")
```

```
C-index score for c_total: 0.9141907740422205]
C-index score for Cd: 0.8995907629348144]
C-index score for Pd: 0.8744519146448407]
```

Results for Leave-Replicas-Out cross-validation

In [9]:
```
#In this cell run your script for leave-Replicas-Out cross-validation and print the corresponding results.


#Get the dataframe with duplicates
yd = map_replicates(y)


lrocv(X, y["c_total"].to_numpy(), yd, 3, "c_total")
lrocv(X, y["Cd"].to_numpy(), yd, 3, "Cd")
lrocv(X, y["Pb"].to_numpy(), yd, 3, "Pb")
```

```
C-index score for c_total: 0.8186734427938493]
C-index score for Cd: 0.7614523739925669]
C-index score for Pb: 0.7689480937069362]
```

Interpretation of results

Answer the following questions based on the results obtained

- Which cross-validation approach had more optimistic results?
- Which cross-validation generalize better on unseen data? Why?

In this cell write your answers to the questions about Interpretation of Results.

Which cross-validation approach had more optimistic results?
-Leave-one-out had better c-score.

Which cross-validation generalize better on unseen data? Why?
-I think that since leave-replicas-out removes the most similiar data points, it doesn't have as much data points close to itself during training as loocv, therefore forcing it to be able to predict better on unknown datapoints. With leave-one-out there is data leakage present in the form of replicas to the current one-fold as those are not independent, and we have already trained on similar data points, making the prediction of the last replica easier.